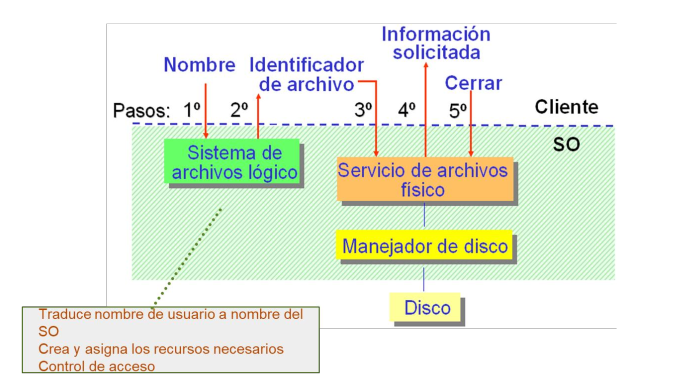
**TEMA 4: GESTIÓN DE ARCHIVOS**

1. **INTERFAZ DEL SISTEMA DE ARCHIVOS**

**FUNCIONAMIENTO**

Al hacer un open sobre un archivo, los mecanismos relacionados con la cuestión de privacidad, comprueban los permisos que tiene, de manera que cuando se genera un descriptor de archivos, se establece un mecanismo de seguridad, y si se intenta llevar a cabo una acción de la que no se tiene permiso, el SO alertará.

Una vez se pasan los controles de acceso,  se recibe un descriptor (que es un entero, no un puntero a un archivo, es un índice a una tabla, por lo que no da acceso directo). Una vez que se tiene el descriptor, se realizan las operaciones y se cierra el archivo. Esta semántica permite separar los sistemas de archivos lógicos del sistema de archivo físico. Los sistemas de archivos están en máquinas distintas.



La elección de las funciones de la API no es neutra, los diseñadores optaron por favorecer los tipos de operaciones más frecuentes en detrimento de las menos frecuentes.

* Optimizar caminos frecuentes – menos llamadas al sistema
* Favorecer accesos secuenciales

La API (interfaz) de los archivos no es independiente de la semántica. Cuando se leen n bytes de un archivo, el puntero de lectura/escritura se va a p + n. El acceso secuencial está más optimizado porque es más frecuente y por tanto necesito menos llamadas al sistema, la lectura está más optimizada que la escritura.

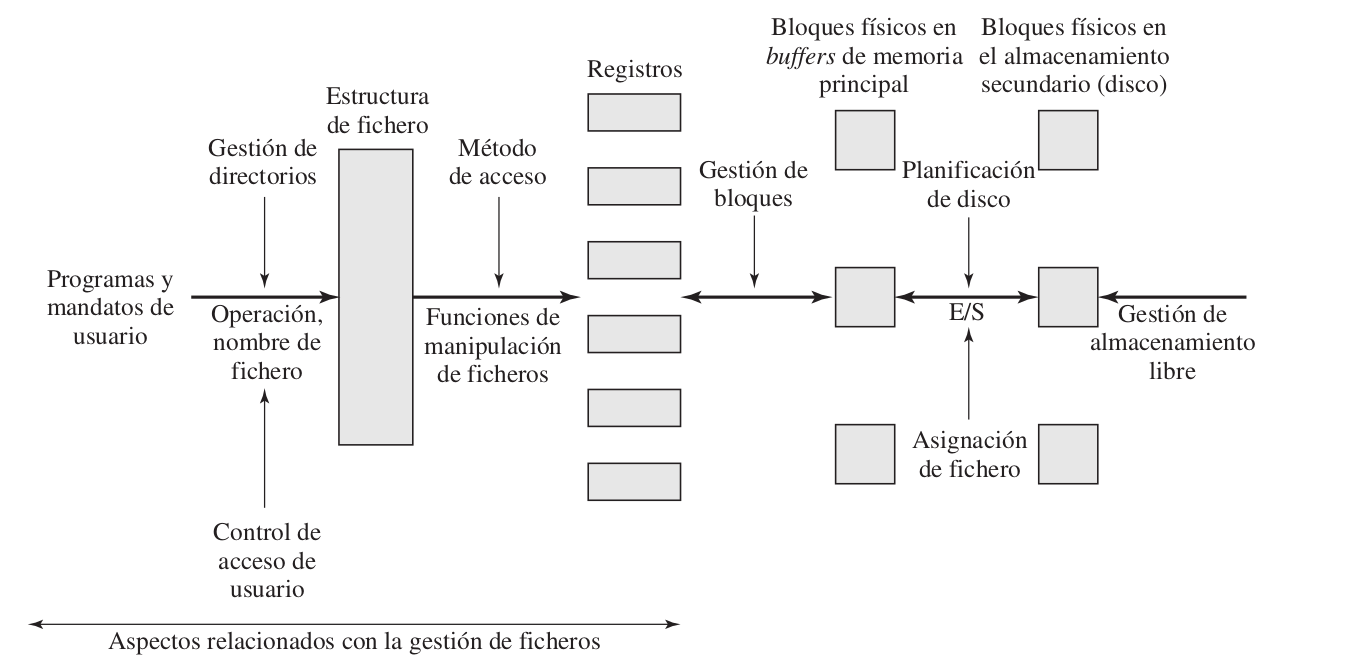
**ATRIBUTOS DE ARCHIVOS**

Los atributos (metadatos) de un archivo son los datos que mantiene el SO para describir el mismo, es decir implementan la abstracción del archivo. Algunos de estos **atributos** son el nombre, el tipo, la ubicación, el tamaño, la protección, los tiempos de creación, la modificación y el último acceso.

**PSEUDO-SISTEMAS DE ARCHIVOS**

La interfaz para manipular archivos se ha mostrado muy potente y flexible. Una idea sería utilizar esta interfaz para acceder a datos o recursos del SO que no son realmente sistemas de archivos (con soporte permanente) (Ejemplos: /proc, /dev, o /devfs).

**ESTRUCTURA DE CAPAS**



**BÚFER DE E/S**

Un **búfer** es una memoria intermedia de almacenamiento temporal que se utiliza para mejorar el rendimiento.

¿Cómo mejoro el rendimiento de E/S?

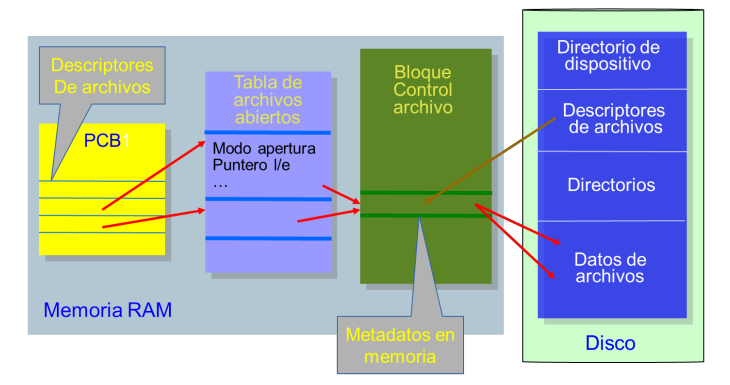
* Acoplando velocidades entre CPU y dispositivos.
* Diferenciando tamaños de datos transferidos.
* Minimizando el tiempo en que un proceso está bloqueado en espera de una operación L/E.
* Desacoplando la E/S de gestión de memoria.

Los búferes se usan a nivel de SO y de bibliotecas.

**Caché de disco** es el conjunto de búferes de E/S de disco gestionados por un algoritmo de reemplazo.

**2. DISEÑO**

**GESTIÓN DE MEMORIA**



**GESTIÓN DEL ALMACENAMIENTO SECUNDARIO**

Cuando se construye un SA, se establece el método de asignación de archivos, es decir,  cómo se van a almacenar los archivos, indicando el formato que se le quiera dar y se crean las estructuras de datos correspondientes.

Al formatear un disco, se borran los datos, pero de forma indirecta.

En almacenamiento secundario, un fichero está compuesto por una colección de bloques. El sistema operativo o sistema de gestión de ficheros es responsable de asignar bloques a los ficheros. Esto supone dos aspectos relacionados con la gestión. Primero se debe asignar espacio de almacenamiento secundario a los ficheros y segundo, es necesario guardar una traza del espacio disponible para su asignación. Estas dos tareas están relacionadas, existiendo una interacción entre la estructura de los ficheros y las políticas de asignación.

Métodos de asignación de ficheros:

* Asignación contigua
* Asignación encadenada o enlazada
* Asignación indexada

Gestión del espacio libre:

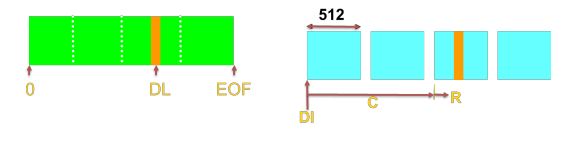
* Tabla de bits
* Porciones libres encadenadas
* Indexación
* Lista de bloques libres

1. **MÉTODOS DE ASIGNACIÓN DE FICHEROS**

**i) ASIGNACIÓN CONTINUA**

La correspondencia de una Dirección Lógica (DL) a una Física:

* DL / Tam. Bloque = Cociente (C) y Resto (R)
* Bloque a acceder = C + Dirección\_inicio
* Desplazamiento en bloque = R



Se asigna un único conjunto contiguo de bloques.

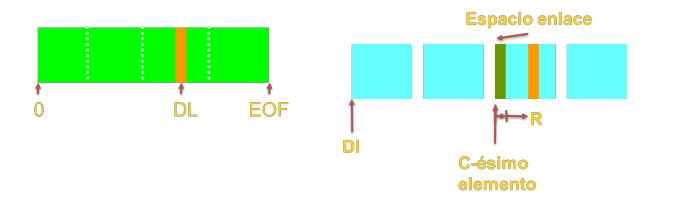
**Ventajas**: es sencillo, sólo se necesita la ubicación de comienzo (número de bloque) y su longitud. Acceso directo y rápido.

**Desventajas:** La asignación es dinámica, por lo que se produce fragmentación de disco. Los archivos no pueden aumentar salvo que se realice compactación (es ineficiente). Si tengo un archivo que ocupa 2 bloques y lo aumento de tamaño, tengo que buscar 3 huecos libres y se me quedan los dos anteriores vacíos.

**Ejemplo:** En almacenamiento para swap se utiliza la asignación continua, por eso tiene una partición independiente. Para el área de la memoria se toma el doble de la memoria RAM para reducir la fragmentación.

**ii) ASIGNACIÓN ENLAZADA**

Correspondencia: DL = C y R

* Bloque a acceder en C-ésimo elemento de la lista.
* Desplazamiento en el bloque = R +1

Se asignan bloques conforme se necesitan y se enlazan en una lista. El almacenamiento en disco es una lista de bloques donde cada bloque tiene un puntero al siguiente bloque. Se almacena donde empieza el archivo y un puntero al siguiente bloque, de tal forma que se puede recuperar el archivo de forma encadenada. En el propio bloque se guarda el puntero.

Permite el crecimiento dinámico, creando bloques. El acceso aleatorio no es el más adecuado. Para acceder al último hay que leer todos los anteriores. Los punteros consumen el espacio de almacenamiento. Si algún puntero de la lista se borra se podría perder parte del archivo.

**Ventajas**:

* Simple (sólo se necesita dirección de inicio).
* Crecimiento dinámico sin fragmentación.

**Inconvenientes**:

* Acceso aleatorio ineficiente.
* Los punteros consumen espacio →  Mejora : agrupación de bloques (clúster).
* Problema de seguridad por pérdida de punteros. Solución: lista doble enlazada → sobrecarga.

FAT (File Allocation Table)

El sistema de archivos FAT se compone de cuatro secciones:

* El sector de arranque. Siempre es el primer sector de la partición (volumen) e incluye información básica, punteros a las demás secciones, y la dirección de la rutina de arranque del sistema operativo.
* La región FAT. Contiene dos copias de la tabla de asignación de archivos (por motivos de seguridad). Estos son mapas de la partición, indicando qué clústeres están ocupados por los archivos.
* La región del directorio raíz. Es el índice principal de carpetas y archivos.
* La región de datos. Es el lugar donde se almacena el contenido de archivos y carpetas. Por tanto, ocupa casi toda la partición. El tamaño de cualquier archivo o carpeta puede ser ampliado siempre que queden suficientes clusters libres. Cada cluster está enlazado con el siguiente mediante un puntero. Si un determinado cluster no se ocupa por completo, su espacio remanente se desperdicia.

Una partición se divide en un conjunto de clusters de idéntico tamaño. Son pequeños bloques discontinuos. El tamaño del clúster depende de la variante de FAT utilizada. Varía entre 2 y 32 kilobytes. Cada archivo ocupa uno o más clusters en función de su tamaño. De manera que un archivo queda representado por una cadena secuencial de clusters (una lista enlazada). Cada clúster de la cadena no tiene por qué ser adyacente al anterior. Esto es lo que provoca la fragmentación.

La tabla de asignación de archivos consta de una lista de entradas. Cada entrada contiene información sobre un clúster:

* La dirección del siguiente clúster en la cadena.
* Si es pertinente, la indicación de "fin de archivo" (que es también el fin de la cadena).
* Un carácter especial para indicar que el clúster es defectuoso.
* Un carácter especial para indicar que el clúster está reservado (es decir, ocupado por un archivo).
* El número cero para indicar que el clúster está libre (puede ser usado por un archivo).
* El tamaño de estas entradas también depende de la variante FAT en uso: FAT16 usa entradas de 16 bits, FAT32 usa entradas de 32 bits, etc

FAT (File Allocation Table): se trata de una variante del método enlazado de Microsoft que actualmente se suele usar en pendrives y disco extraíbles.

Cada partición reserva un espacio para la FAT que contiene una entrada por bloque de disco y está indexada por el número de bloque. Hay una copia de la FAT en caché para reducir el tiempo de búsqueda.

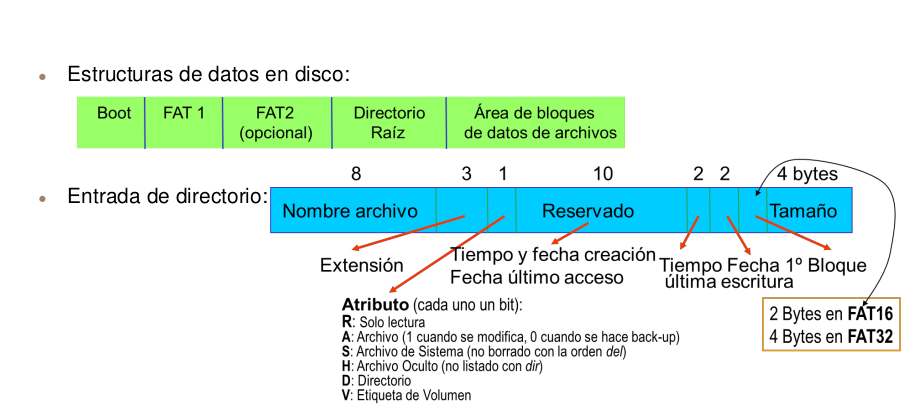
Una aproximación de la lista enlazada es la que utiliza el SA FAT, que se sigue utilizando a día de hoy en los pendrives y discos extraíbles. La diferencia con el enlazado es que queda la lista enlazada pero la saco de los datos y me la llevo a una tabla. En el disco construyo una reserva de espacio que es la tabla de asignación y en dicha tabla construyo la lista de bloques. En la tabla tengo una entrada por bloque, por lo que hay al menos tantas entradas como bloques de datos. En resumen, la lista la tengo una tabla, tiene entradas como bloques y las listas las construyo ahí.

**Ventaja**: cuando se arranca el sistema se lleva a memoria la FAT. La lectura no se hace sobre disco sino sobre la memoria que es más eficiente. No se desperdicia espacio, se ajusta la FAT al tamaño de disco. El problema es que se manipula la FAT en memoria, si se va la luz hay problemas, ya que se pierden las modificaciones que hayamos hecho en la FAT.

En una entrada a directorio: FAT 32 (punteros más grandes), FAT 16 (punteros más pequeños), cuanto más punteros más bloques, con 16 bits, 2¹⁶ bloques . La diferencia es el tamaño de los punteros,

De los 4 bytes para el tamaño tan solo se utilizan 31 bits para direccionar el tamaño del archivo. Hoy en día se solventa partiendo el archivo.

En reservado están los atributos de un archivo, que es una orden muy frecuente.

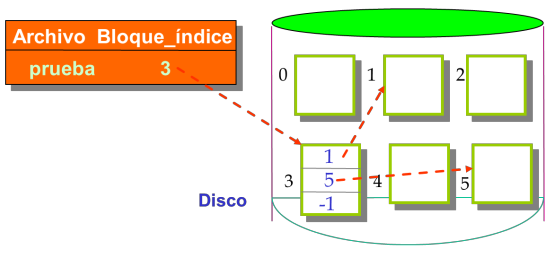
Se intenta optimizar el camino más frecuente; es decir, las órdenes más frecuentes. Ejemplo: buscar un archivo, ls…

**iii)** **ASIGNACIÓN INDEXADA**

Voy a tener un bloque de índices, que es un bloque entero de punteros.

Se necesitan uno o varios bloques que se usan como tabla de índices a bloques de datos por archivo.

La tabla de asignación de ficheros contiene un índice separado de un nivel por cada fichero; el índice tiene una entrada por cada porción asignada al fichero. La asignación puede realizarse mediante bloques de tamaño fijo (se elimina la fragmentación externa) o porciones de tamaño variable (mejora la proximidad). Da soporte tanto a acceso secuencial como directo a los ficheros, por lo que es la forma de asignación de ficheros.más usada.



**Ventajas**:

* Acceso aleatorio es rápido y no hay fragmentación.
* Acceso dinámico sin fragmentación, pero tiene la sobrecarga del bloque de índices.

**Desventajas**:

* Posible desperdicio de espacio en el bloque de índices
* Tamaño del bloque de índices. Soluciones  (por ejemplo): UNIX utiliza bloques de índices enlazados multinivel.

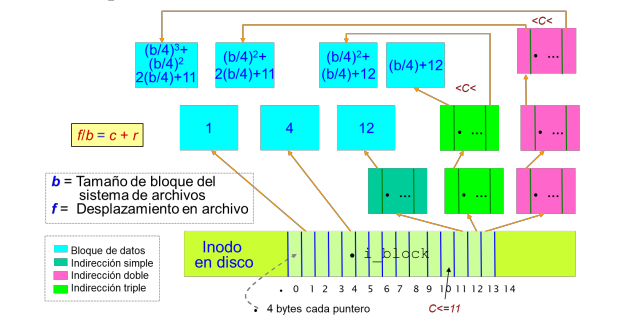
INODO (NODO ÍNDICE)

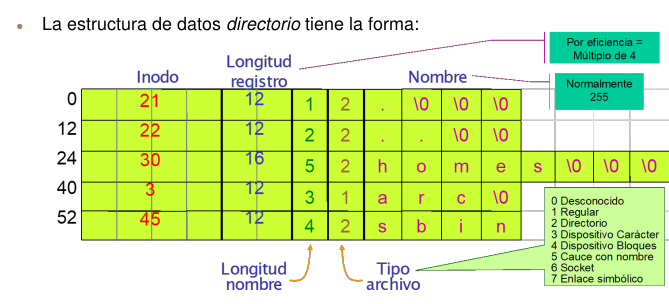
Es una estructura de control que contiene la información clave necesaria de un fichero particular para el sistema operativo. Varios nombres de ficheros se pueden asociar con un único inodo, pero un inodo activo se asocia con exactamente un fichero, y cada fichero es controlado por exactamente un inodo. Para los archivos pequeños se usa direccionamiento directo, que son los que más trabajo y para los grandes hasta 3 niveles.  En Linux se usa direccionamiento múltiple.



**Sistemas ext2**

INODO: DIRECCIONAMIENTO DE BLOQUES DE DATOS



La asignación de ficheros se realiza a nivel de bloque. La asignación es dinámica, es decir, cuando se necesita, en lugar de utilizar preasignación. Por tanto, los bloques de un fichero en disco no son necesariamente contiguos. Se utiliza un método indexado para guardar traza de cada fichero, con parte del índice almacenado en el nodo-i del fichero.

DIRECTORIO DE LINUX

Un directorio es simplemente un fichero que contiene una lista de nombres de ficheros más punteros a inodos asociados.

ÁREA DE INTERCAMBIO

El uso del área de intercambio depende de los algoritmos de gestión de memoria:

* Intercambio: procesos completos.
* Paginación: páginas sucias de procesos.

Lugar para la asignación de espacio en disco:

* Un archivo del sistema de archivos (Windows) - Fácil de implementar pero ineficiente (fragmentación).
* Partición de disco independiente (Linux, en general) - No utiliza estructura de directorios ni sistema de archivos, asignación contigua (un slot por página). Eficiente.

En Linux el intercambio se hace sobre una partición independiente, en Windows sobre el SA. Linux también permite un archivo. El acceso a través del SA es más costoso que acceder a una partición, la ventaja es que puede crecer y decrecer con la necesidad que quiera, mientras que la partición no puede, habría que reparticionar el sistema.

1. **GESTIÓN DEL ESPACIO LIBRE**

De la misma forma que se asigna el espacio a los ficheros, también se debe gestionar el espacio que no está actualmente asignado a ningún fichero. Para llevar a cabo cualquiera de las técnicas de asignación de ficheros descritas previamente, es necesario saber qué bloques están disponibles en el disco. Por tanto, se necesita una tabla de asignación de disco además de la tabla de asignación de ficheros.

Para ello se puede utilizar una tabla de bits, los bloques vacíos a 0 y los bloques en archivos a 1, porciones libres encadenadas, indexación y lista de bloques libres.

* Tabla de bits: este método utiliza un vector que está formado por un bit por cada bloque en el disco. Cada entrada 0 corresponde a un bloque libre y cada 1 corresponde a un bloque en uso.
* Porciones libres encadenadas: Las porciones libres se pueden encadenar utilizando un puntero y valor de longitud en cada porción libre. Este método tiene una sobrecarga de espacio insignificante, porque no se necesita una tabla de asignación de disco, sino simplemente un puntero al comienzo de la cadena y la longitud de la primera porción. Este método es apropiado para todos los métodos de asignación de ficheros.
* Indexación: trata el espacio libre como un fichero y utiliza una tabla de índices tal y como se describió en la asignación de ficheros.
* Lista de bloques libres: a cada bloque se le asigna un número secuencialmente y la lista de los números de todos los bloques libres se mantiene en una porción reservada del disco.

**RAID**

**RAID (Redundant Array of Inexpensive Disks)**: consta de varios discos funcionando como una unidad donde cada bloque se descompone en sub-bloques que son almacenados en discos separados. El esquema RAID consta de siete niveles, del cero al seis. Estos niveles no implican una relación jerárquica, sino que designan a diversas arquitecturas de diseño que comparten tres características comunes:

1. RAID corresponde con un conjunto de unidades físicas de disco tratado por el sistema operativo como un único dispositivo lógico.

2. Los datos están distribuidos a lo largo de las unidades físicas de un vector.

3. La capacidad de redundancia del disco se utiliza para almacenar información de paridad, que garantiza que los datos se pueden recuperar en caso de que falle un disco.

Las técnicas de despiece de datos mejoran la tolerancia a fallos y realizan la transferencia y posicionamiento en paralelo.

**PLANIFICACIÓN DE DISCO**

Se utiliza cuando tenemos un disco electromagnético. La operación más costosa es fisionar el cabezal en la pipa correspondiente y esperar que pase por debajo. Cuando el usuario pide escribir o leer del disco, el SO no atiende las peticiones en el orden en que se producen sino buscando el menor movimiento de disco, la distancia de posicionamiento (reducir el tiempo de posicionamiento del cabezal sobre el bloque de datos). Hay diferentes algoritmos que se confeccionan en el arranque. En linux encontramos: noop, deadline y cfq.

**3. IMPLEMENTACIÓN**

**VFS (Sistema de Archivos Virtual)**

Para manipular diferentes sistemas de archivos con las mismas órdenes, surge el sistema de archivo virtual, que es una capa de abstracción de software superior, con las mismas llamadas para cualquier sistema subyacente. Es la capa software entre el usuario y los diferentes tipos de SA que suministra al usuario/proceso una interfaz única para el acceso a archivos independientes del tipo de SA.

El Sistema de Archivos Virtual (VFS) permite soportar diferentes tipos de sistemas de archivos (SA).

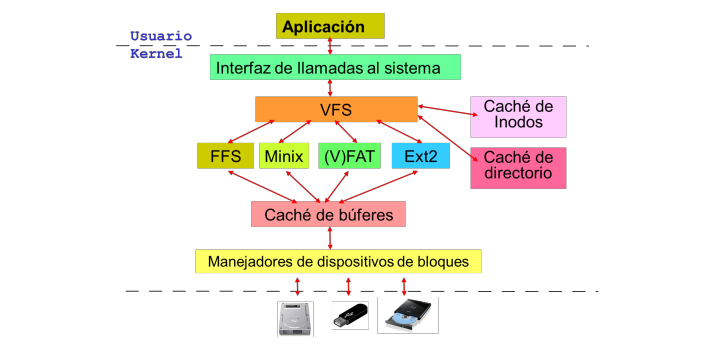
Ejemplo: podemos ver como las operaciones para acceder a archivos son independientes del tipo de sistema de archivos.

SA SOPORTADOS

Los sistemas soportados por VFS podemos agruparlos en:

* Basados en disco → sistemas que gestionan una partición de disco (Ejemplo:  Ext2, ms-dos, VFAT, NTFS, CDROM, FFS, …).
* Basados en red →  sistemas de archivos en red como NFS, SMB, AFS, …
* Especiales →  no gestionan disco sino que son la interfaz a otro tipo de objetos, como son /proc, /dev/pts.

Cualquier sistema, una vez montado, tiene la misma forma de acceso.

**VFS: ESTRUCTURA**

**VFS: MODELO DE ARCHIVO COMÚN**

Definir a nivel de VFS una estructura de datos capaz de almacenar cualquier otro dato de un archivo  en un archivo común suficientemente genérico.

Construye un modelo de archivo común capaz de representar todos los archivos de los diferentes sistemas de archivos soportados. Un sistema de archivos específico debe traducir su organización física a la del modelo de archivo común de VFS.

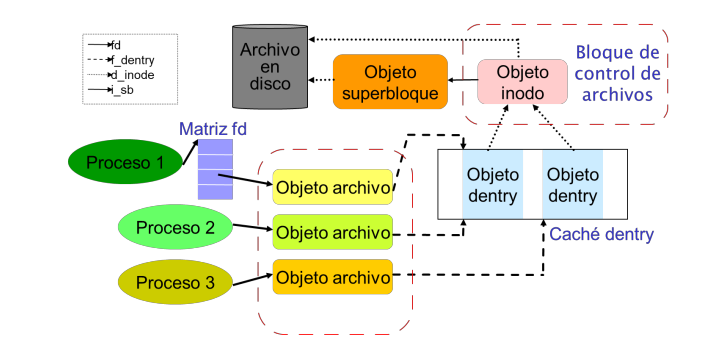
Un modelo archivo común utiliza el modelo objeto (construcción software que define los datos y los métodos para operar sobre ellos). El archivo común es un objeto que contiene punteros a las funciones que implementan los métodos sobre una implementación concreta de sistema de archivos.

**VFS:OBJETOS EN MEMORIA**

VFS es un esquema orientado a objetos. Estos me van a permitir manipular los archivos. Las estructuras de datos en memoria que usa VFS son:

* Objeto superbloque →  almacena información de un sistema de archivos montado. Se corresponde con el directorio de dispositivo.
* Objeto inodo →  almacena información de un sistema de archivos específico (bloque de control de archivos).
* Objeto archivo →  describe la interacción entre un archivo abierto y un proceso (tabla de archivos abiertos).
* Objeto dentry (entrada de directorio) →  enlace entre una entrada de directorio y el archivo correspondiente.

VFS: RELACIONES (EJEMPLO)



Hay 3 archivos abiertos. Tengo 2 objetos dentry apuntando a ese inodo  y tengo un archivo con dos enlaces duros con dos nombres. La traducción en el nombre 1 y en el nombre 2, creo un enlace duro. Cada proceso ha abierto un archivo.

**i) OBJETO SUPERBLOQUE**

Representa un sistema de ficheros montado específico. Almacena información que describe un sistema de ficheros específico. Típicamente, el superbloque corresponde al superbloque del sistema de ficheros o bloque de control del sistema de ficheros, que se almacena en un sector especial en el disco. Tiene como elementos el dispositivo sobre el cual el SA de ficheros está montado, el tamaño de bloque básico, lista de ficheros abiertos, puntero a la raíz…

**ii) OBJETO INODO**

Objeto inodo, o inodo en memoria, es la ED que contiene los metadatos de un archivo en disco además de campos dedicados a la gestión de las caché de inodos (listas de inodos sin usar, inodos en uso, e inodos sucios). Algunos de los campos más relevantes a nuestro estudio son:

* i-list → puntero a la lista de inodos.
* i\_dentry →  punteros a la lista dentry.
* i\_count →  contador de uso.
* i\_op →  operaciones sobre inodos.
* u →  información específica del SA.

**iii) OBJETO ARCHIVO**

Se utiliza para representar un fichero abierto por un proceso. El objeto se crea en respuesta a la llamada al sistema open() y se destruye en respuesta a la llamada al sistema close(), y viene representado por una estructura file. Estos objetos no tienen imagen en disco pues representan una sesión de trabajo sobre un archivo. Los principales campo de este estructura son:

* f\_pos –  Puntero de posición del fichero (posición actual desde la cual tendrá lugar la siguiente operación). Puntero de l/e.
* f\_mode →   r/w, permisos, etc.
* f\_op →  tabla de operaciones de archivos.
* f\_count →  contador de uso, != 0 si está en uso.
* Información para mantener los objetos en diferentes listas (identificador del usuario, identificador de grupo del usuario…).

TRADUCCIONES DE NOMBRES

Hay que traducir el nombre de un archivo al nombre interno que va a ser el que el sistema operativo corresponde con el inodo.

Los sistemas Unix usan dos espacios de nombres para archivos:

* Espacio de nombres del SO →  inodos.
* Espacio de nombres de usuario →  pathname

El kernel debe establecer la correspondencia (traducir) entre los nombres de usuario y sus correspondientes inodos. Esta traducción es lenta y en algunos casos compleja (enlaces y puntos de montaje).

**iv) OBJETO DENTRY**

El objeto dentry (entrada de directorio) asocia el componente del pathname con su inodo asociado. Cuando VFS lee en memoria una entrada de directorio este la transforma en objeto dentry. Hay un objeto dentry por cada componente de pathname que busca un proceso (el componente puede ser un nombre de directorio o un nombre de fichero). Se usan como caché para acelerar la traducción de nombres de archivos. Algunos campos son:

* d\_name →  nombre en la entrada
* d\_inode →  inodo del archivo
* d\_mounts →  dentry de la raíz de un FS (*SA?*)  montado.
* d\_covers →  dentry de un punto de montaje

NOTA: Si padre e hijo comparten el mismo archivo abierto, si alguno de ellos escribe en dicho archivo, los dos ven las modificaciones. Si escribe el padre y luego el hijo, el hijo escribirá a partir de donde haya dejado el padre el puntero offset. Para cualesquiera procesos, cuando comparto un archivo el SA no controla la concurrencia del acceso, el resultado dependerá de cómo se escriban los archivos en él.